Kaggle Name: anthonyfong123

Play Prediction:

For this task I used the homework 3 problem 4 model as my base. Jaccard and most popular were the same, but this time I included a weight with each method. I also filtered the entire dataset to create dictionaries for every user. One dictionary had each user and their total amount of transformed hours, another had every user and the length of all their reviews combined normalized by the longest review, and the last had every user and the amount of times they had early access to games. I used grid search to discover the best thresholds for every model and kept track of all the times the models passed the threshold. From there I used grid search to find the optimal weights for each method, and it seems that when either Jaccard or most popular predicts played, then I should predict played. Though if Jaccard and most popular predicted not played, then all 3 dictionaries would have to predict played for me to still predict play. My method was in essence a form of AdaBoost. The idea that by combining a lot of decent classifiers I could make a good one while still having a quick training model. I also tried different similarity formulas such as Cosine, but Jaccard gave the best results. I tried incorporating other machine learning models like Logistic Regression and Random Forest as well, but in the end the described method above gave the best results.

Category Prediction:

In category prediction I also used homework 3 as my initial setup. I filtered all stop words based on the English language and made the dictionary size 5000 words. This model gave me a decent score on the public leaderboard, but I thought a 5000 word dictionary might be overfitting a bit. As such I moved on to my final model which was built using TF-IDF. I once again filtered for all English stop words, but I also set sublinear_tf to be true in the TF-IDF function I was using. What subliner_tf did was that it scaled the term frequency in the TF-IDF model by assigning a weight generated by taking the logarithm of the term frequency. I decided to scale the term frequencies in this way, because I did not believe that merely counting the occurrences of a term truly represented its impact on the document. A term that appears 20 times does not necessarily hold 20 times the weight of a term that appears 1 time. I tried hyper tuning the TF-IDF function parameters such as stripping accents and generating different ngrams, but the described model above gave the best results. I used the sklearn feature extraction TF-IDF vectorizer for my function. All parameters not specifically mentioned were left as default.